

Переменные

- Создание переменной резервирует место, или пространство в памяти для хранения значений.
- Компилятору необходимо, чтобы вы указали тип данных для каждой объявляемой переменной.
- С++ предлагает большой ассортимент встроенных типов данных.

Типы данных C++

- Целочисленный тип, встроенный тип, представляет собой целое число. Для определения переменной целочисленного типа используется ключевое слово `int`.
- C++ требует чтобы вы указали тип и идентификатор для каждой переменной.
- Идентификатор это имя для переменной, функции, класса, модуля, или чего-либо другого определенного пользователем. Идентификатор начинается с буквы (A-Z или a-z) или нижнего подчеркивания (`_`), с последующими дополнительно буквами, нижними подчеркиваниями, и цифрами (от 0 до 9).

Переменные

- Например, определим переменную под названием myVariable которая может хранить целочисленные значения:

```
main.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int myVariable = 10;
7      cout << myVariable;
8      return 0;
9  }
```

- Различные операционные системы могут резервировать разные размеры памяти для одного и того же типа данных.

Переменные

- Определяйте все переменные с именем и типом данных до их использования в программе. В случае, если у вас есть несколько переменных одинакового типа, можно определять их в одном объявлении, разделяя их запятыми.
- Переменным могут быть присвоены значения и они могут использоваться для выполнения операций.
- Например, мы можем дополнительно создать переменную `sum`, и сложить две переменные.
- Используйте оператор `+` для сложения двух чисел.
- Давайте создадим программу для подсчета и вывода на экран двух целочисленных переменных.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a = 30;
7      int b = 12;
8      int sum = a + b;
9
10     cout << sum;
11
12     return 0;
13 }
14
```

Переменные

- Всегда помните, что все переменные должны быть определены с именем и типом данных до того, как они будут использованы.
- Следующая программа подсказывает пользователю ввести число и сохраняет его в переменной a:

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a;
7      cout << "Please enter a number \n";
8      cin >> a;
9
10     cout << a;
11
12     return 0;
13 }
14
```

Переменные

- После запуска программы выводится сообщение "Please enter a number", затем ожидается ввод пользователем числа и нажатие кнопки Enter, или Return.
- Введенное число сохраняется в переменной a.
- Программа будет ждать столько времени, сколько необходимо пользователю чтобы ввести число.
- Вы можете выполнить в консоли столько раз, сколько необходимо, как сделано в следующей программе:

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a, b;
7      cout << "Enter a number \n";
8      cin >> a;
9      cout << "Enter another number \n";
10     cin >> b;
11
12     cout << a << " " << b;
13
14     return 0;
15 }
```

Переменные

- Давайте создадим программу, которая позволяет ввести два числа и выводит на экран их сумму.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a, b;
7      int sum;
8      cout << "Enter a number \n";
9      cin >> a;
10     cout << "Enter another number \n";
11     cin >> b;
12     sum = a + b;
13     cout << "Sum is: " << sum << endl;
14
15     return 0;
16 }
```

Переменные

- Указывать тип данных необходимо лишь один раз, во время объявления переменной.
- После этого переменная может быть использована без ссылки на тип данных.

```
int a;  
a=10;
```

- Указание типа данных для переменной более одного раза вызовет синтаксическую ошибку.
- Значение переменной может быть изменено столько раз, сколько это необходимо в программе.

main.cpp

```
1  #include <iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6      int a = 100;  
7      a = 50;  
8      cout << a;  
9      return 0;  
10 }
```


Арифметические операции

- С++ поддерживает следующие арифметические операторы.
- Вы можете использовать несколько арифметических операторов в одной строке.

Оператор	Символ	Вид
Сложение	+	$x + y$
Вычитание	-	$x - y$
Умножение	*	$x * y$
Деление	/	x / y
Деление по модулю	%	$x \% y$

Арифметические операции

- Оператор вычитания вычитает один операнд из другого.
- Оператор умножения перемножает операнды
- Оператор деления делит первый операнд на второй. Любой остаток отбрасывается для возвращения целочисленного значения.
- Если один или оба операнда являются значениями с плавающей точкой, то оператор деления осуществляет деление с плавающей точкой.
- Деление на 0 сломает вашу программу.
- Оператор деления по модулю (%) неофициально известен как оператор остатка, потому что он возвращает остаток после деления целочисленных переменных.
- Приоритет операторов определяет порядок вычисления, который влияет на то, как выражения будут вычислены. Определенные операторы имеют приоритет выше других; например, оператор умножения имеет приоритет выше, чем у оператора сложения.
- Скобки присваивают операциям высокий приоритет. Если выражение в скобках находится в другом выражении, также закрытом скобками, то сперва вычисляется выражение, лежащее внутри.

Операторы

- Если никакие выражения не заключены в скобки, то мультипликативные (умножение, деление, деление по модулю) операторы будут вычислены до аддитивных (сложение, вычитание) операторов.
- Простой оператор присваивания (=) присваивает правую часть выражения к левой части.
- C++ имеет короткие операторы умножения, деления и присваивания

```
int x = 10;  
x += 4; // equivalent to x = x + 4  
x -= 5; // equivalent to x = x - 5
```

- Оператор присваивания (=) присваивает правую часть выражения к левой части.
- Такой же простой синтаксис применим для операторов умножения, деления и деления по модулю.

```
x *= 3; // equivalent to x = x * 3  
x /= 2; // equivalent to x = x / 2  
x %= 4; // equivalent to x = x % 4
```

Операторы

- Оператор инкремента используется для увеличения целочисленного значения на единицу.
- `x++`; //equivalent to `x = x + 1`
- Оператор инкремента используется для увеличения целочисленного значения на единицу.
- Оператор инкремента имеет две формы, префиксную и постфиксную.
- `++x`; // префикс
- `x++`; // постфикс
- Префикс увеличивает значение, а затем высчитывает выражение.

Операторы

- Постфикс высчитывает выражение, а затем осуществляет увеличение.
- `x = 5;`
- `y = ++x;`
- `// x is 6, y is 6`
- `x = 5;`
- `y = x++;`
- `// x is 6, y is 5`
- Пример префиксной формы увеличивает значение `x`, а затем присваивает его к `y`.
- Пример постфиксной формы присваивает значение `x` к `y`, а затем производит его увеличение.

Операторы

- Оператор декремента (--) работает почти таким же образом, как и оператор инкремента, но вместо увеличения значения, он уменьшает его на единицу.
- --x; // префикс
- x--; // постфикс

Условные выражения

- Выражение `if` используется для выполнения некоторого кода, при соответствии условиям.

```
if (condition) {  
  statements  
}
```

- Условие определяет, какое выражение будет выполнено. Если условие истинно, то выражение в фигурных скобках будет выполнено.
- Если условие ложно, то выражение просто игнорируется, и программа продолжает выполнение после тела оператора `if`.

Условные выражения

- Используйте операторы отношения для работы с условиями.
- Оператор if проверяет условие ($7 > 4$), убеждается, что оно истинно, и затем выполняет оператор cout.
- Если мы поменяем оператор отношения с "больше" на "меньше" ($7 < 4$), то выражение не будет выполнено и ничего не будет выведено на экран.
- Условие, указанное в операторе if, не требует точки с запятой.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      if (7 > 4) {
7          cout << "Yes";
8      }
9
10     return 0;
11 }
```


Дополнительные операторы отношения:

Оператор	Описание	Пример
\geq	Больше или равно	$7 \geq 4$ Истина
\leq	Меньше или равно	$7 \leq 4$ Ложь
$==$	Равно	$7 == 4$ Ложь
$!=$	Не равно	$7 != 4$ Истина

Операторы

- Оператор не равно вычисляет операнды, определяет равны ли они друг другу. Если операнды не равны, то условие считается истинным.

```
main.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      if (10 != 10) {
7          cout << "Yes";
8      }
9
10     return 0;
11 }
```

Операторы

- Вышеописанное условие считается ложным и блок кода не будет выполнен.
- Вы можете использовать операторы отношения, чтобы сравнивать переменные внутри оператора if.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a = 55;
7      int b = 33;
8      if (a > b) {
9          cout << "a is greater than b";
10     }
11
12     return 0;
13 }
14
```

Операторы

- Оператор if может использоваться вместе с оператором else, который выполняется, когда условие ложно.

```
if (condition) {  
statements  
}  
else {  
statements  
}
```

Операторы

- Код проверит условие:
- Если оно истинно, то код внутри выражения if будет выполнен.
- Если оно ложно, то будет выполнен код внутри выражения else.
- Если используется только одно выражение внутри оператора if/else, то фигурные скобки могут быть опущены

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int mark = 90;
7
8      if (mark < 50) {
9          cout << "You failed." << endl;
10     }
11     else {
12         cout << "You passed." << endl;
13     }
14
15     return 0;
16 }
```

Операторы

- Во всех предыдущих примерах использовалось только одно выражение внутри оператора if/else, но вы можете включить столько выражений, сколько вам необходимо.
- Вы также можете использовать вложенный оператор if (внутри другого оператора if).
- C++ обеспечивает опцией неограниченного использования вложенных операторов if/else.
- Помните, что все операторы else должны иметь соответствующий оператор if.
- В операторе if/else, одиночное выражение может быть включено без заключения в фигурные скобки.
- Включение в фигурные скобки в любом случае является хорошей практикой, так как оно вносит ясность в код и улучшает его читаемость.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int mark = 90;
7
8      if (mark < 50) {
9          cout << "You failed." << endl;
10         cout << "Sorry" << endl;
11     }
12     else {
13         cout << "Congratulations!" << endl;
14         cout << "You passed." << endl;
15         cout << "You are awesome!" << endl;
16     }
17
18     return 0;
19 }
```

задание

- Написать программу, которая будет решать дискриминант. Исходные данные для расчёта дискриминанта взять самостоятельно и определить статически (в самом коде). Продемонстрировать все ветки решения программы (когда D больше, меньше и когда $= 0$), заменяя исходные значения a , b и c .
- К каждому примеру сделать скриншот работы консольного окна..
- Создать отчет по данной работе.

- *Не повторяйте исходные значения своих одногруппников, во избежание недоразумений при сдаче отчета.

Отчет (Фамилия_№лабораторной.doc)

- Титульный лист

Тема работы: «Работа с переменными и операторами».

- Отчет

Цель работы: Научиться работе с переменными и операторами.

Задачи: 1. Написать программу, которая будет рассчитывать дискриминант, исходные данные переменным задать статически. (например: $a = 2$, $b = 35$, $c = 8$) 2. Продемонстрировать все условия работы программы: а) $D > 0$ б) $D < 0$ в) $D = 0$. D^* - дискриминант.

Ход работы.

<Ваша работа, шаги, рисунки>

Вывод.

*Исходные данные для переменных a , b , c – определяйте так, чтобы у каждой подгруппы они были свои.